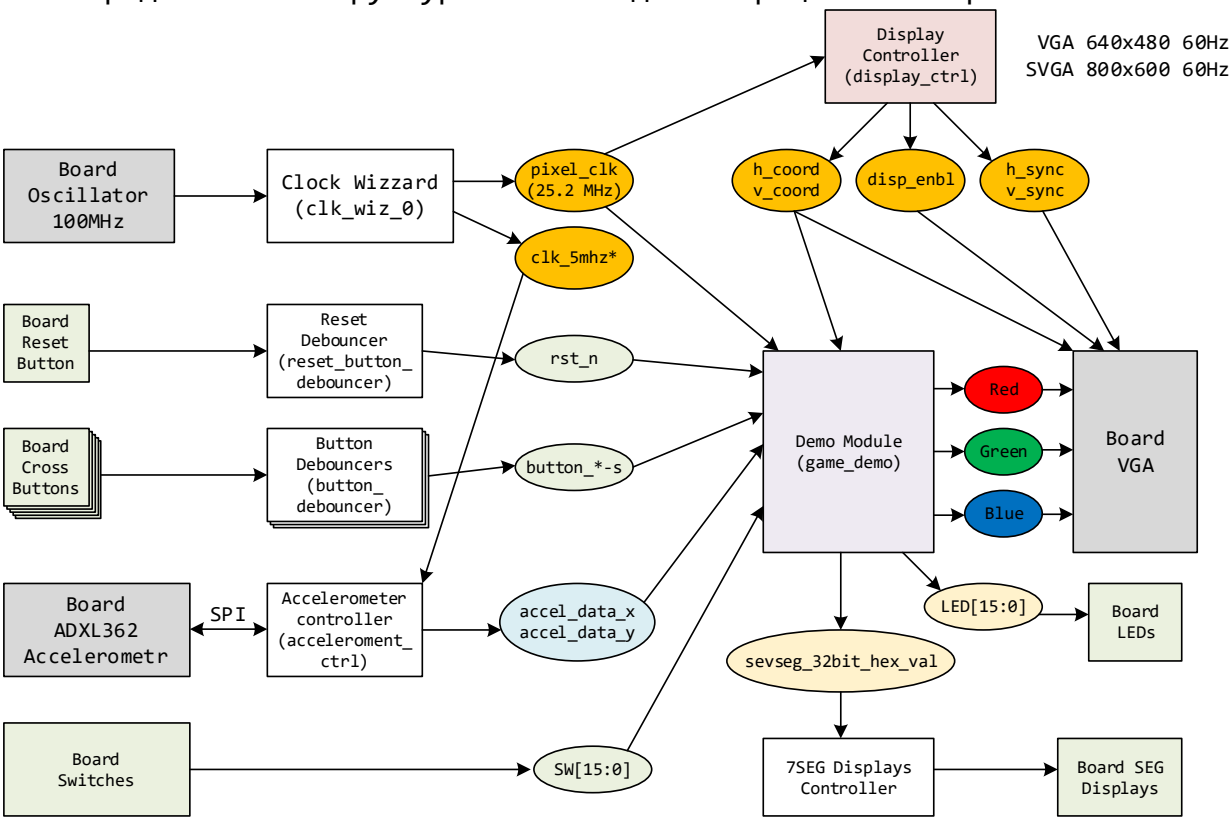
Описание демонстрационного проекта

Два проекта: sberday_nexys_a7 и sberday_nexys_a7_svga

Изначально, планировалась одна версия демонстрационного проекта. Однако, после закупки оборудования, было определено, что мониторы плохо работают с разрешением 640x480 (дублируют и размывают изображение). Поэтому задание было переработано на более высокое разрешение 800x600(SVGA). По сути структура проекта осталась одинаковой, за исключением учета цифр нового разрешения. Ключевое отличие заключается в том, что при разрешении 640x480 для управления логикой отрисовки используется тактовый сигнал с частотой в 25.2 МГц, а в разрешении SVGA, уже 40 МГц, что в свою очередь увеличивает требования к быстродействию разработанного дизайна, или потребует применить знания о переходах между тактовыми доменами. Мы рекомендуем начать разработку на базе проекта в разрешении SVGA, и если вы столкнетесь с проблемой закрытия таймингов, рассмотреть вариант миграции на более низкое разрешение.

Описание демонстрационного FPGA проекта

Ниже представлена структурная схема демонстрационного проекта.



Структурная схема демонстрационного проекта.

- **Clock Wizzard**

  Vivado IP блок из IP Catalog (см. Инструкция Vivado). Используется для формирования частоты пикселей 40 МГц в версии SVGA (25.2 МГц для VGA), используемых для генерации сигналов в контроллере дисплея и демонстрационном модуле.
  Также формирует два тактовых сигнала clk_5mhz_0d и clk_5mhz_180d, используемые для работы контроллера акселерометра общающемся по протоколу SPI с микросхемой акселерометра ADXL362.

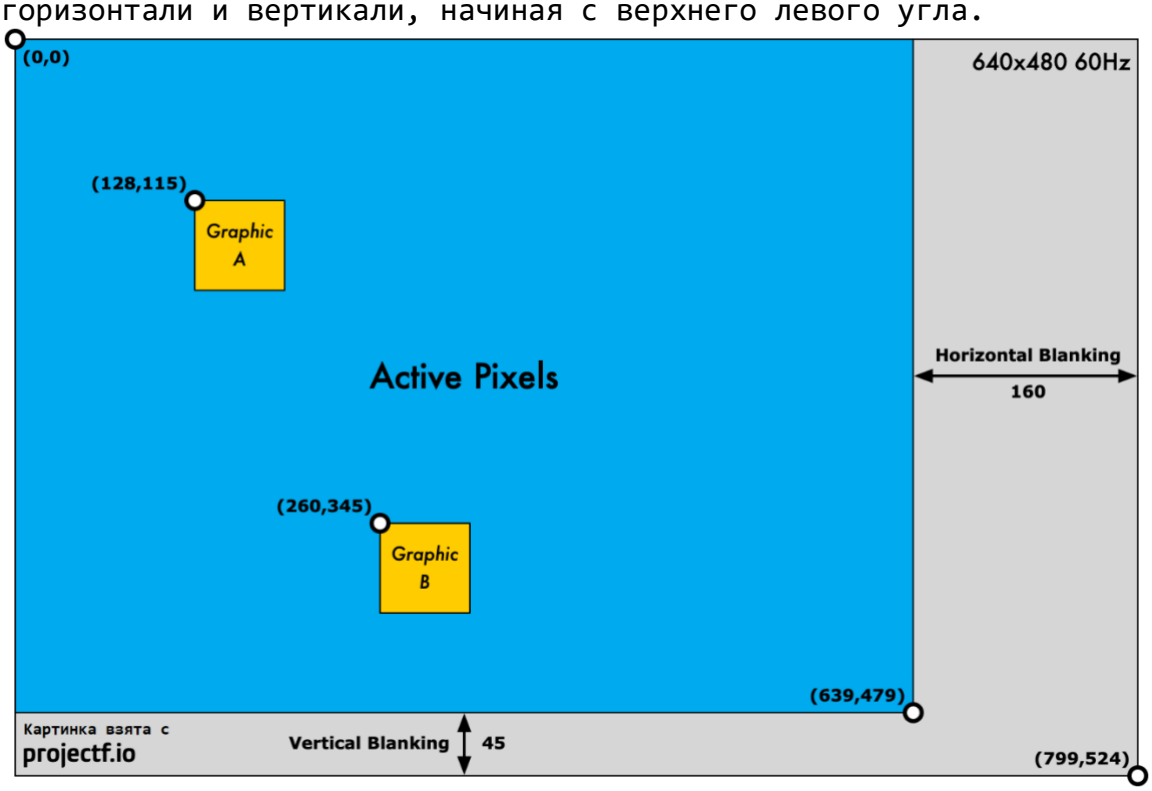
```
64   clk_wiz_0 clk_gen (
65     .clk_in1  ( CLK100MHZ       ),
66     .resetn   ( pll_rst_n       ),
67     .clk_out1 ( pixel_clk       ), // 40 MHz
68     .clk_out2 ( clk_5mhz0d      ),
69     .clk_out3 ( clk_5mhz180d    ),
70     .locked   ( pll_locked      )
71   );
```

- **Reset & Buttons Debouncers**

  Кнопки – один из самых старых и популярных интерфейсов человек-машина. Когда пользователь нажимает на кнопку, она замыкает электрическую цепь, позволяя току течь через неё. Это замыкание может быть использовано для включения, выключения или управления другим устройством, в нашем случае FPGA. Механические кнопки могут создавать дребезг при нажатии, что выражается в скачках напряжения. Для того чтобы избавить сигнал от дребезга используются специальные схемы, называемые дебаунсеры, их задача анализировать сигнал, подаваемый с кнопки и отсекать изменения сигнала, длина которых меньше определенного времени фильтрации. У кнопки Reset и других кнопок различная полярность. **Рекомендуется не изменять данные модули и использовать только сигналы, полученные после фильтрации.**

```
73   reset_button_debouncer reset (
74     .clk_100mhz   ( CLK100MHZ  ),
75     .clk_5mhz     ( clk_5mhz0d ),
76     .reset_button ( CPU_RESETN ),
77     .pll_rst_n    ( pll_rst_n  ),
78     .pll_locked   ( pll_locked ),
79     .rst_n        ( rst_n      )
80   );
…
85   button_debouncer dbns_center (
86     .clk         ( pixel_clk  ),
87     .arst_n      ( rst_n      ),
88     .sw_i        ( BTNC       ),
89     .sw_state_o  ( button_c   ),
90     .sw_down_o   ( button_c_d ),
91     .sw_up_o     ( button_c_u )
92   );
```

- **Display Controller**
  Контроллер дисплея генерируется сигналы **h_coord** и **v_coord**, показывающие положение пикселя, которое обрисовывается сейчас по горизонтали и вертикали, начиная с верхнего левого угла.



Принцип отрисовки изображения.

Помимо координат, контроллер дисплея генерирует сигналы вертикальной и горизонтальной синхронизации **h_sync** и **v_sync**, передаваемые внешнему дисплею вместе с значениями пикселей, а также сигнал **disp_enbl** показывающий активную область пикселей.

```
198  display_ctrl display_ctrl (
199    .pixel_clk  ( pixel_clk ), // Pixel clock 40 MHz
200    .rst_n      ( rst_n     ),
201    .h_sync     ( h_sync    ), // horizontal sync signal
202    .v_sync     ( v_sync    ), // vertical sync signal
203    .disp_enbl  ( disp_enbl ),
204    .h_coord    ( h_coord   ), // horizontal pixel coord
205    .v_coord    ( v_coord   )  // vertical pixel coord
206  );
```

- **Board Switches**
  Помимо кнопок на плате существуют еще 16 переключателей (Switches). Их значения доступны в верхнем модуле через сигналы **SW[15:0]**. Демонстрационные проект использует крайние правые три переключателя **SW[2:0]** для выбора цвета заднего фона.

```
5    input   logic  [15:0]    SW        ,
```

- **Board LEDs & 7SEG displays**
  Для вывода информации с платы помимо VGA дисплея вам доступны светодиоды и 8 семисегментных индикаторов.
  На плате присутствует 16 зеленых светодиодов и 2 RGB светодиода, доступных по имени LED.

```
7    output  logic  [15:0]    LED       ,
8    output  logic            LED16_R, LED16_G, LED16_B,
9    output  logic            LED17_R, LED17_G, LED17_B,
```

  В демонстрационном проекте используются только зеленые светодиоды. Первые два показывают текущий режим демонстрации. 3 светодиод не используется, 4-9 светодиоды показывают нажатие кнопок крестика, верхние 7 светодиодов, показывают состояние верхних 6 переключателей.

```
126  assign LED [15:8] = SW[15:8];
127  assign LED [7:3] = {button_u, button_l, button_r, /
     button_d, button_c};
128  assign LED [1:0] = demo_regime_status;
129  //RGB
130  assign {LED16_R, LED16_G, LED16_B} = 3'b000;
131  assign {LED17_R, LED17_G, LED17_B} = 3'b000;
```

  Для работы семисегментных индикаторов используется специальный контроллер (sevseg_ctrl), которые управляет катодами и общим анодом дисплеев. Для пользователя доступно 32 бита данных, по 4 бита на каждый дисплей, выводящие значения в HEX формате (от 0 до F). В демонстрационном проекте используются 4 из 8 дисплеев, 2 крайних правых показываю значение Y акселерометра, 5 и 6 дисплей показываются значения X акселерометра.

```
141 sevseg_ctrl seven_segments_controller (
142  .clk                 (digital_clock_divider [12]  ),
143  .arst_n              (rst_n                       ),
144  .sevseg_32bit_hex_val (sevseg_32bit_hex_val       ),
145  .seg_active_n        ({CG, CF, CE, CD, CC, CB, CA}),
146  .don_active_n        (DP                          ),
147  .anodes              (AN                          )
148 );
149 assign sevseg_32bit_hex_val = {8'b0,
                                   accel_x_end_of_frame,
                                   8'b0,
                                   accel_y_end_of_frame};
```

- **Accelerometer Controller**
  На плате присутствует 3-х осевой гироскоп ADXL362, общающийся с платой по протоколу SPI. Контроллер акселерометра реализует простой SPI Master порт, который после конфигурации гироскопа, читает восьмибитные значения осей X и Y, но не читает ось Z. В одном из режимов демонстрационного проекта, эти значения используются для управления нашим объектом.

```
151  accelerometr_ctrl accelerometr_ctrl (
152    //Clocks & Resets
153      .spi_clk            ( clk_5mhz0d    ),
154      .spi_clk_out        ( clk_5mhz180d  ),
155      .arst_n             ( rst_n         ),
156    //Accelerometer values
157      .accel_data_x       ( accel_data_x  ),
158      .accel_data_y       ( accel_data_y  ),
159    //Accelerometer IOs
160      .ACL_CSN            ( ACL_CSN       ),
161      .ACL_SCLK           ( ACL_SCLK      ),
162      .ACL_MISO           ( ACL_MISO      ),
163      .ACL_MOSI           ( ACL_MOSI      )
164  );
```

- **Game Demo**
  Демонстрационный модуль Game Demo отвечает за то, что и в каком цвете будет отрисовываться на дисплее. Модуль использует интерфейсы ввода и вывода платы.

  Демонстрационный модуль выводит логотип компании – организатора хакатона. После чего показывает демонстрационный объект, в нашем случае это прямоугольник. У объекта есть свои характеристики:
  - В первую очередь это координаты по горизонтали и вертикали **object_h_coord** и **object_v_coord**.
  - Помимо этого, объект имеет ширину (**object_width**) и высоту (**object_height**). Не стоит забывать, что отрисовка в модуле идет от верхнего левого угла.
  - Еще одной из характеристик объекта, которые мы рекомендуем ввести – это скорость объекта по горизонтали и вертикали (**object_h_speed** и **object_v_speed**). Помимо того, что использование скорости объекта позволит подрегулировать скорость перемещения объекта по экрану. Использовании отрицательных значений скорости позволит изменять направление движения неуправляемых объектов.
  - Сигнал object_draw показывает принадлежность текущего пикселя, который отрисовывает контроллер дисплея, к объекту, и используется для управления цветами RGB.

Одинокий стик ждет ваших указаний.

```
//game.sv
52   parameter     object_width  = 4  ;
53   parameter     object_height = 10 ;
54   logic         object_draw        ;
55   logic [9:0]   object_h_coord     ;
56   logic [9:0]   object_v_coord     ;
57   logic [9:0]   object_h_speed     ;
58   logic [9:0]   object_v_speed     ;
```

В демонстрации есть два режима, переключение между которыми происходит по нажатию кнопки button_c (центральной кнопки крестика).

```
//game.sv
83   always @( posedge pixel_clk ) begin
84     if ( !rst_n ) begin
85       regime_store <= 2'b11;
86     end
87     else if (change_regime && /
                (regime_store == 2'b10)) begin
88       regime_store <= 2'b11;
89     end
90     else if ( change_regime  ) begin
91       regime_store <= regime_store - 1'b1;
92     end
93   end
94   assign change_regime      = button_c    ;
```

В зависимости от режима, стик изменяет положения либо в следствии нажатия кнопок, либо используя значения с гироскопа платы.

Для демонстрации логотипа Сбера используются значения пикселей, которые хранятся в Read Only Memory (ROM). В демонстрационном проекте используется поведенческая модель ROM. Вы можете использовать Vivado IP Блок (см. инструкция Vivado) и преобразовывать картинку из png в сое формат и подгрузить при создании. **Однако, использование блоков ROM, сгенерированных в**

Vivado не позволяет использовать Verilator для отладки поведения проекта без загрузки данных в ПЛИС.

Для того, чтобы Verilator смог отобразить рисунок, демонстрационный проект использует поведенческую модель описания ROM, которая распознается Vivado и Verilator (sber_log_rom.sv).

```
1         module sber_logo_rom (
2           input  wire    [13:0]     addr,
3           output wire    [11:0]     word
4         );
5
6         logic [11:0] rom [(128 * 128)];
7
8         assign word = rom[addr];
9
10        initial begin
11          rom[0    ] = 12'h000;
12          rom[1    ] = 12'h000;
…           …
16395     end
```

В данном случае все значения памяти пословно указаны внутри блока initial begin … end. Данный вариант описания понимается как Verilator, так и Vivado и не требует использования путей до файлов.

Есть альтернативный способ написать поведенческую модель ROM памяти, с помощью функции **$readmemh** и указания файла с содержимым массива памяти. Но тогда необходимо указать путь до файла.

```
1    module sber_logo_rom (
2      input  wire    [13:0]     addr,
3      output wire    [11:0]     word
4    );
5
6    logic [11:0] rom [(128 * 128)];
7
8    assign word = rom[addr];
9
10   initial begin
11     $readmemh ("<your path>/sber_logo.hex", rom);
12   end
13
14   endmodule
```

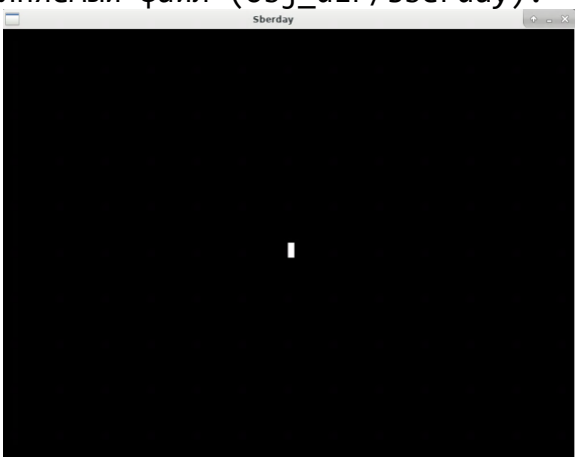
Описание отладки на Verilator

Внутри проекта присутсвует возможность проверить часть функционала вашего модуля без прошивки на FPGA. Для этого используется программа Verilator, преобразующая HDL код в C++ код,а также библиотека SDL для вывода графики.

В папке SIM есть два файла:

- top_sberday.sv – топовый модуль на языке SystemVerilog, отчасти повторяющий топовый модуль FPGA проекта, за исключением части отключенных выводов.
- main_sberday.cpp – тестовое окружение на языке C++, используемое Verilator для отрисовки картинки, а также для управления сигналами объекта top_sberday, созданного Verilator на основе top_sberday.sv

Для формирования исполняемого файла, используйте команду make. После чего запустите исполняемый файл (obj_dir/sberday).



Окно симулятора.

Симулятор работает медленнее чем железо, но позволяет быстрее отслеживать изменения в вашем проекте.

Для выхода из симулятора нажмите Q. Симулятор поддерживает только один демо режим. Для эмуляции кнопок управления используются стрелочки клавиатуры. Для управления цветом заднего фона используются клавиши C,V,B. Данная конфигурация прописана в main_sberday.cpp:

```
99   top->button_u = keyb_state[SDL_SCANCODE_UP];
100  top->button_d = keyb_state[SDL_SCANCODE_DOWN];
101  top->button_r = keyb_state[SDL_SCANCODE_RIGHT];
102  top->button_l = keyb_state[SDL_SCANCODE_LEFT];
103  top->button_c = keyb_state[SDL_SCANCODE_SPACE];
104  top->sw0 = keyb_state[SDL_SCANCODE_C];
105  top->sw1 = keyb_state[SDL_SCANCODE_V];
106  top->sw2 = keyb_state[SDL_SCANCODE_B];
```

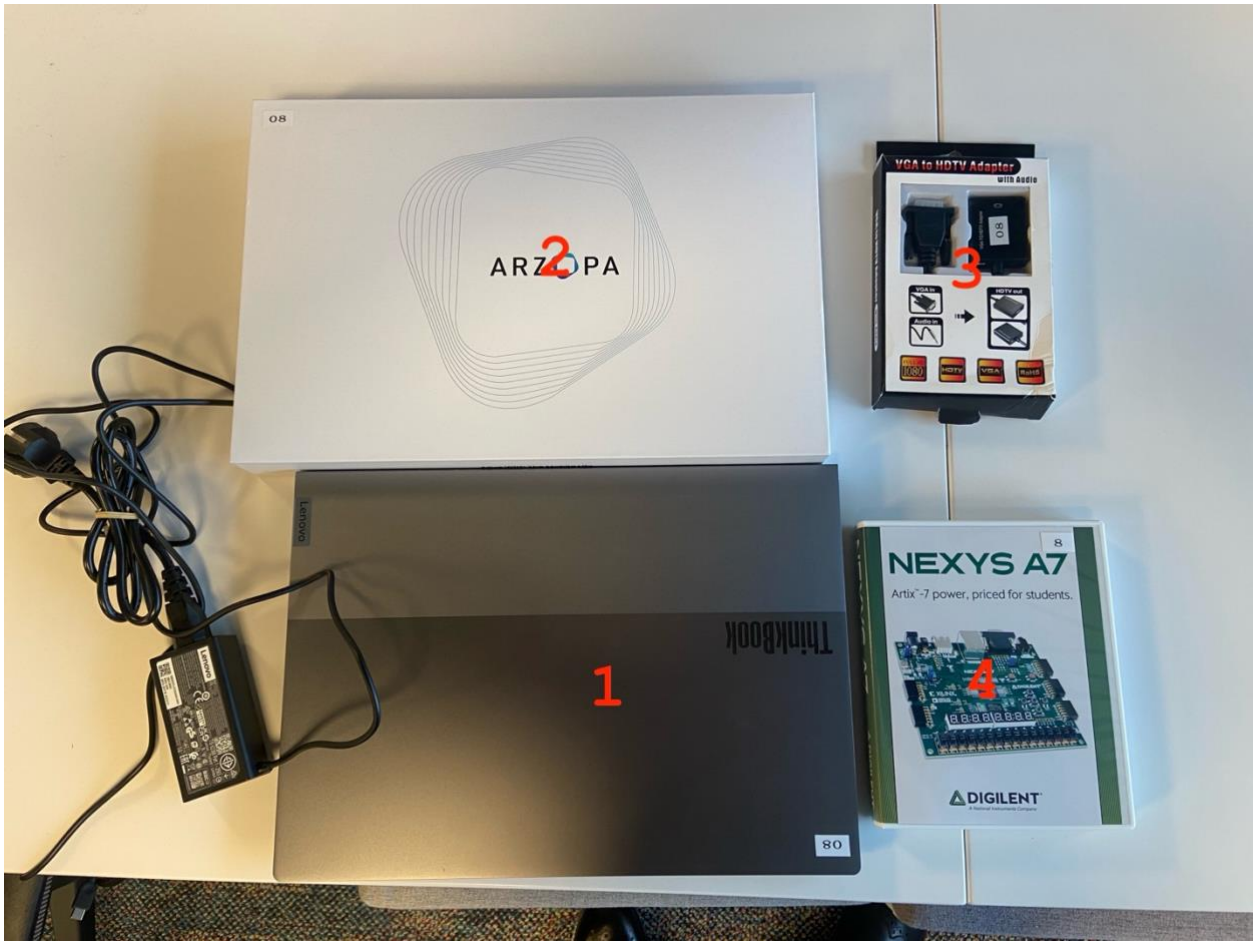
Быстрый старт

1. Сборка стенда

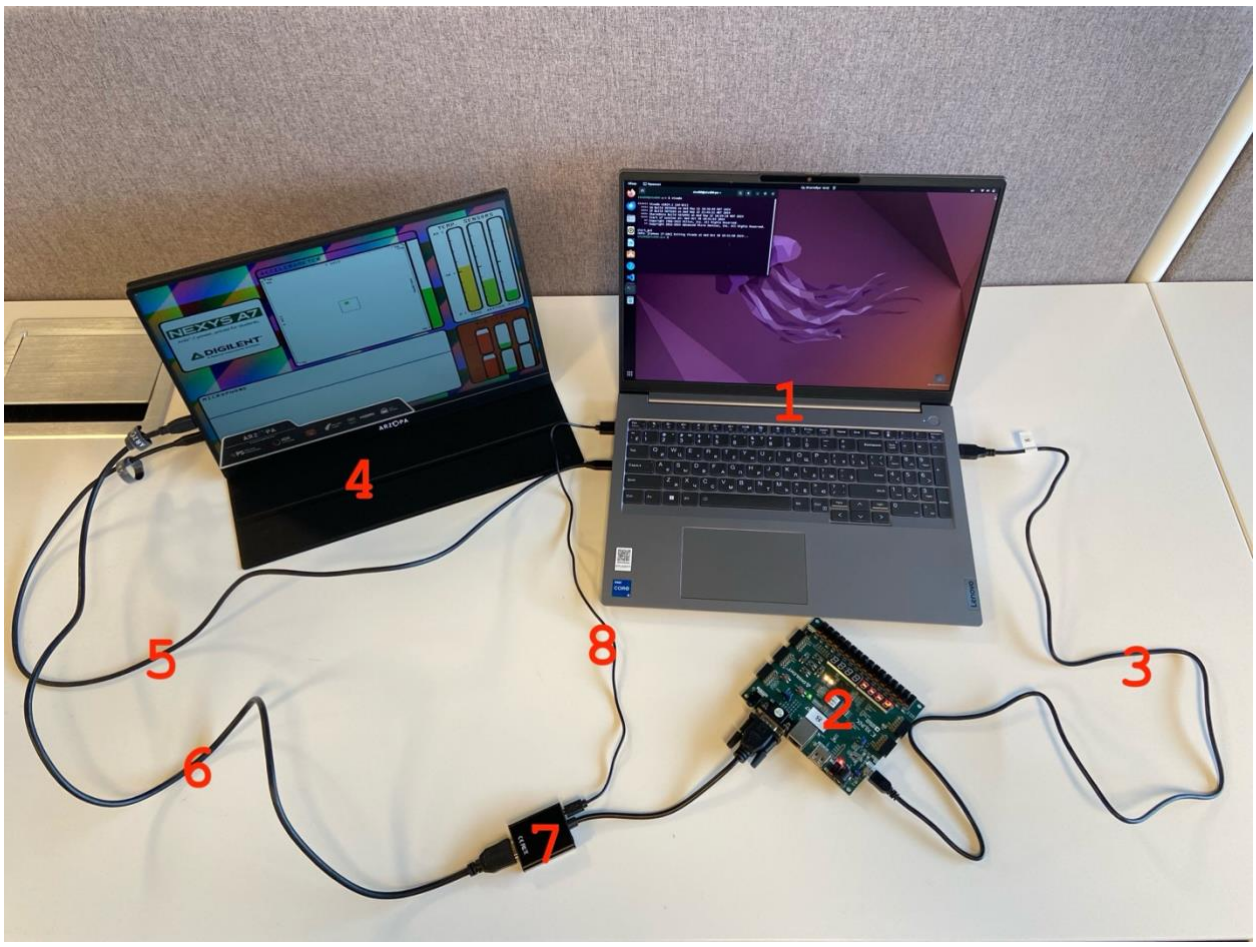
Данный раздел предлагается для более быстрого погружения в работу с предоставленным оборудованием, сборки стенда и проверки работоспособности всех его элементов.

Предоставляемое оборудование включает в себя:

1) Ноутбук с предустановленным необходимым ПО и зарядным устройством;
2) Внешний дисплей с набором кабелей:
   - USB Type C to USB Type C;
   - HDMI to HDMI Mini;
   - USB Type A to USB Type C;
3) Конвертор VGA to HDMI набором кабелей:
   - USB Type A to Micro USB;
   - Mini Jack 3,5 to Mini Jack 3,5;
4) Отладочная плата Digilent Nexys A7 с кабелем Micro USB.



Предоставляемое оборудование.



Стенд для запуска проекта.

Демонстрационный стенд состоит из следующих элементов:

1) Ноутбук;
2) Отладочная плата;
3) Кабель Micro USB из комплекта отладочной платы;
4) Внешний дисплей;
5) Кабель USB Type C to USB Type C из комплекта дисплея;
6) Кабель HDMI to HDMI Mini из комплекта дисплея;
7) Конвертор VGA to HDMI;
8) Кабель USB Type A to Micro USB из комплекта конвертора VGA to HDMI.

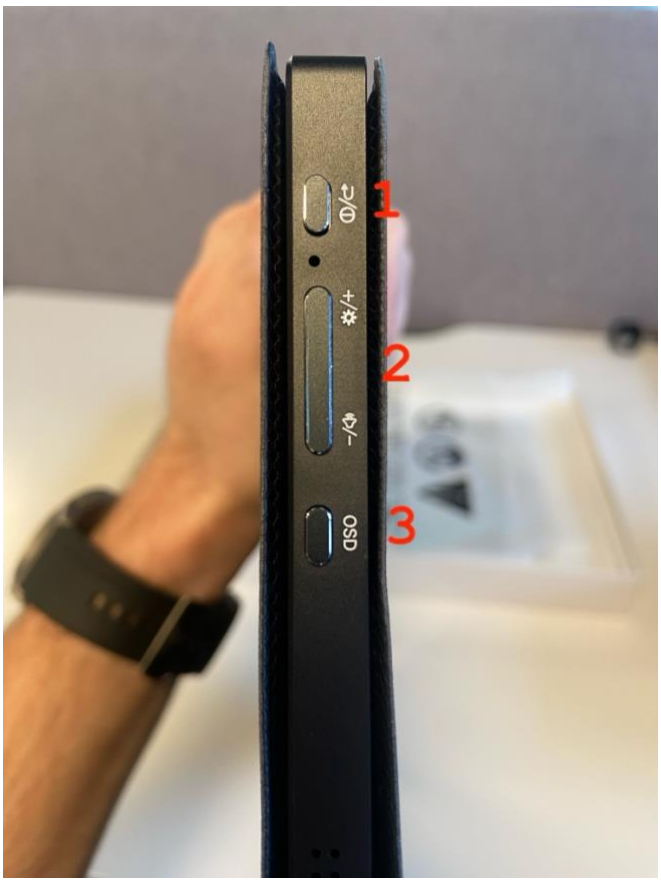
Отладочная плата подключается как показано на рисунке ниже. Необходимо подключить Micro USB кабель к порту 1, а HDMI to VGA конвертор к разъему 2. USB кабель платы подключается к USB порту ноутбука.

Дисплей подключатся кабелем USB Type C to USB Type C к ноутбуку и кабелем HDMI to HDMI Mini к конвертору VGA to HDMI.

Подключение кабелей к отладочной плате.

В память отладочной платы записана демонстрационная прошивка от производителя, можно проверить корректность подключения платы к дисплею. Для того, чтобы включить питание платы надо переключить переключатель 3.

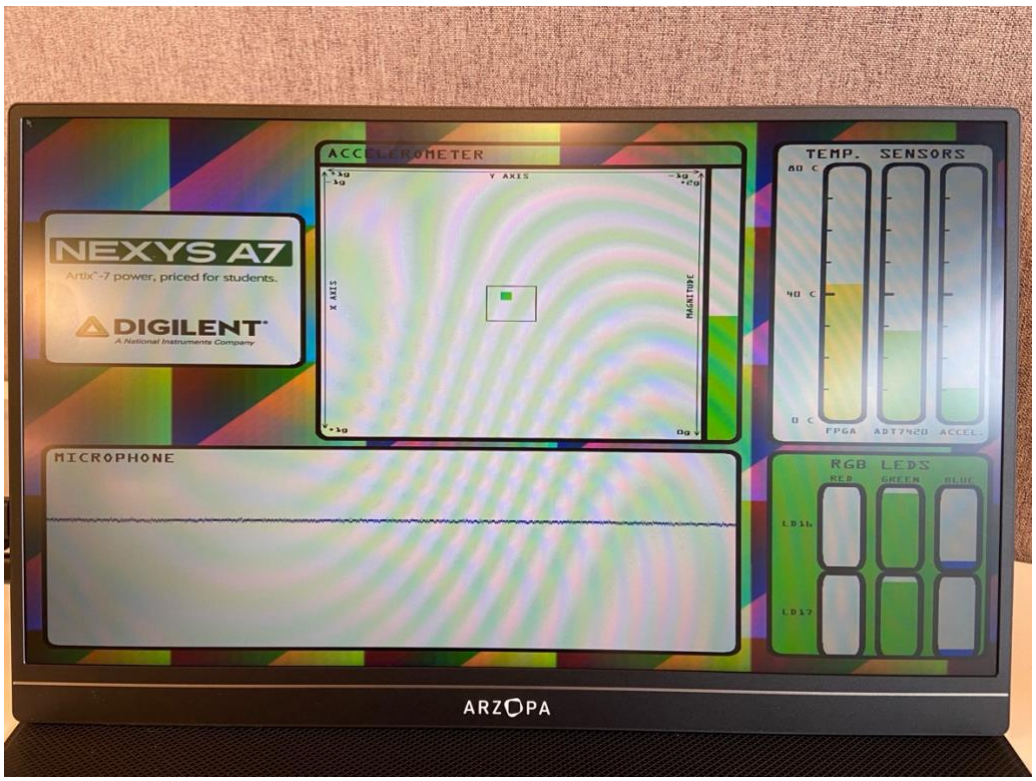
Так как дисплей подключен к ноутбуку через USB, дисплей автоматически перейдет в режим работы второго монитора ноутбука, для воспроизведения изображения, передаваемого из отладочной платы, потребуется переключить источник изображения. Для этого потребуется нажать на 1, после чего появится меню выбора источника изображения, надо выбрать пункт «HDMI».



Обозначение кнопок на внешнем дисплее.

Меню выбора источника изображения.

Далее требуется подтвердить выбор кнопкой 3. Должно появиться изображение как на рисунке ниже.



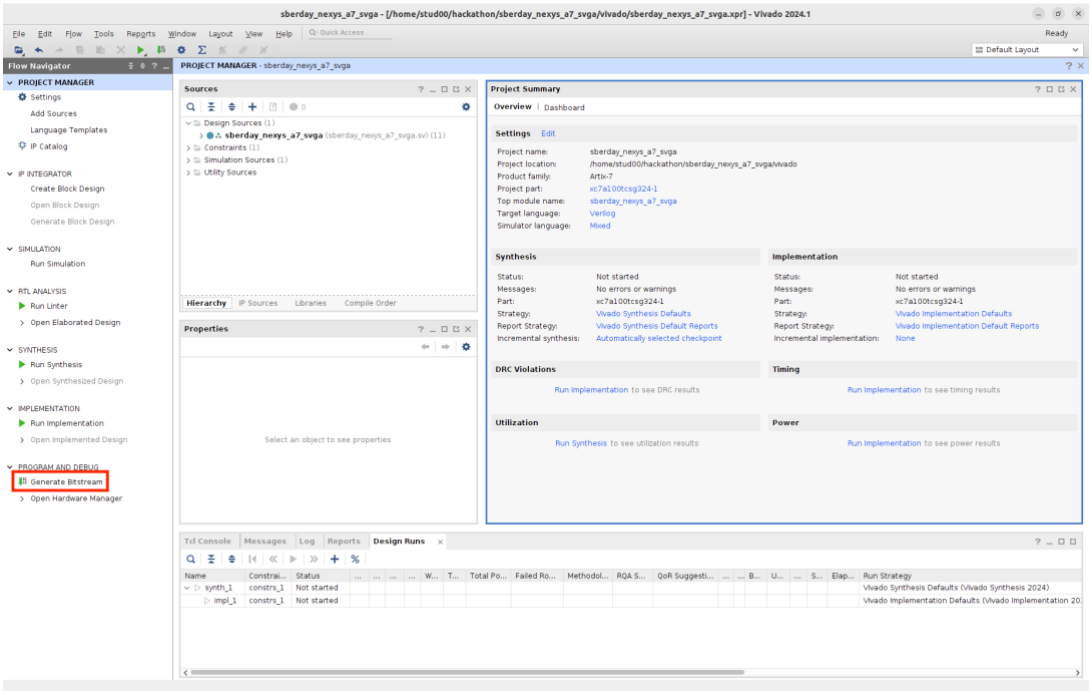
Демонстрационный проект от производителя платы.

Если все получилось, можно переходить к следующему пункту, если же нет, перепроверьте, все ли корректно подключено. В случае возникновения неполадок или вопросов, обращайтесь к организаторам хакатона.

2. Сборка проекта

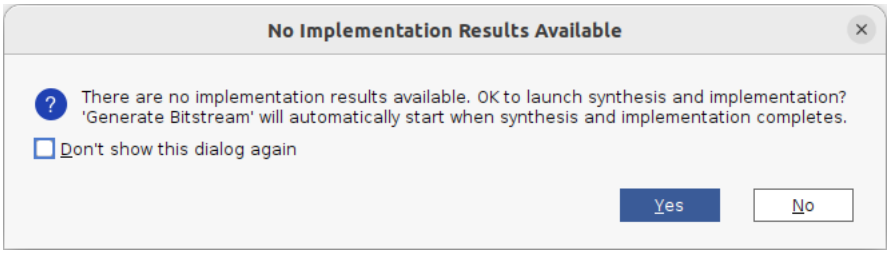
Далее можно собрать предложенный организаторами демонстрационный проект. Для этого откройте терминал на предоставленном ноутбуке и перейдите в папку *«sberday_nexys_a7_svga»* в репозитории с проектом командой:
`cd <repository_dir>/sberday_nexys_a7_svga`

Далее выполните команду `make deploy`, после чего откроется графический интерфейс проекта в Vivado как показано на рисунке ниже. Более подробное описание приведено в инструкции по использованию Vivado (файл 4_Инструкция_Vivado.docx).



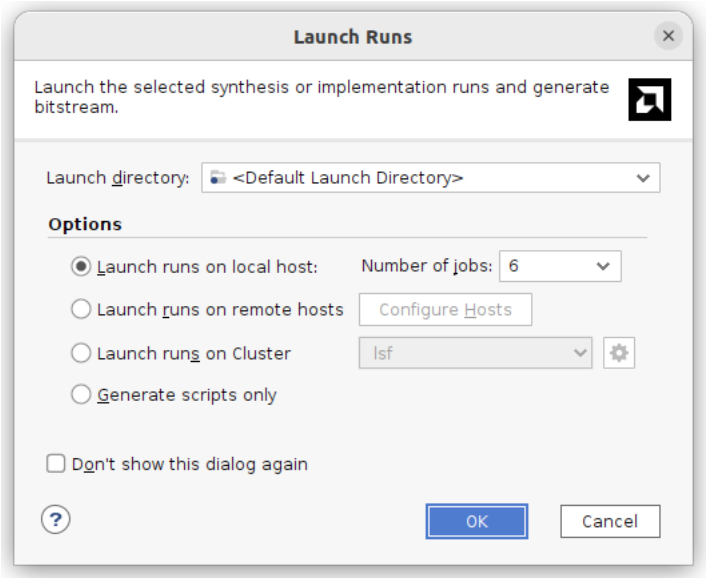
Графический интерфейс Vivado.

Для сборки прошивки нажмите кнопку *«Generate bitstream»*, отмеченную на рисунке выше красным. Vivado предупредит о том, что требуется выполнить синтез и имплементацию прежде чем сгенерировать битстрим и покажет окно, которое приведено на рисунке ниже. Нажмите кнопку *«Yes»*.



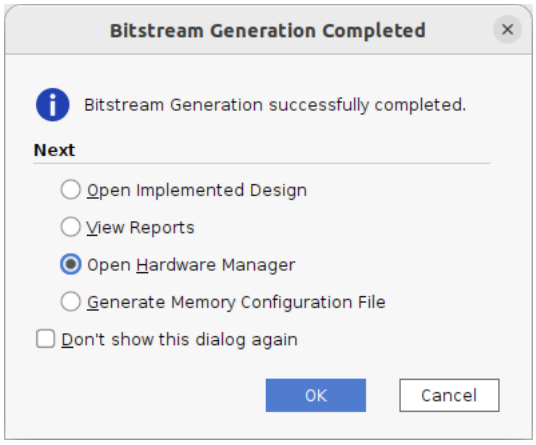
Предупреждение о том, что синтез и имплементация еще не пройдены.

Далее среда предоставит выбор настроек сборки, оставьте их по умолчанию и нажмите *«Ok»*.



Запуск сборки проекта.

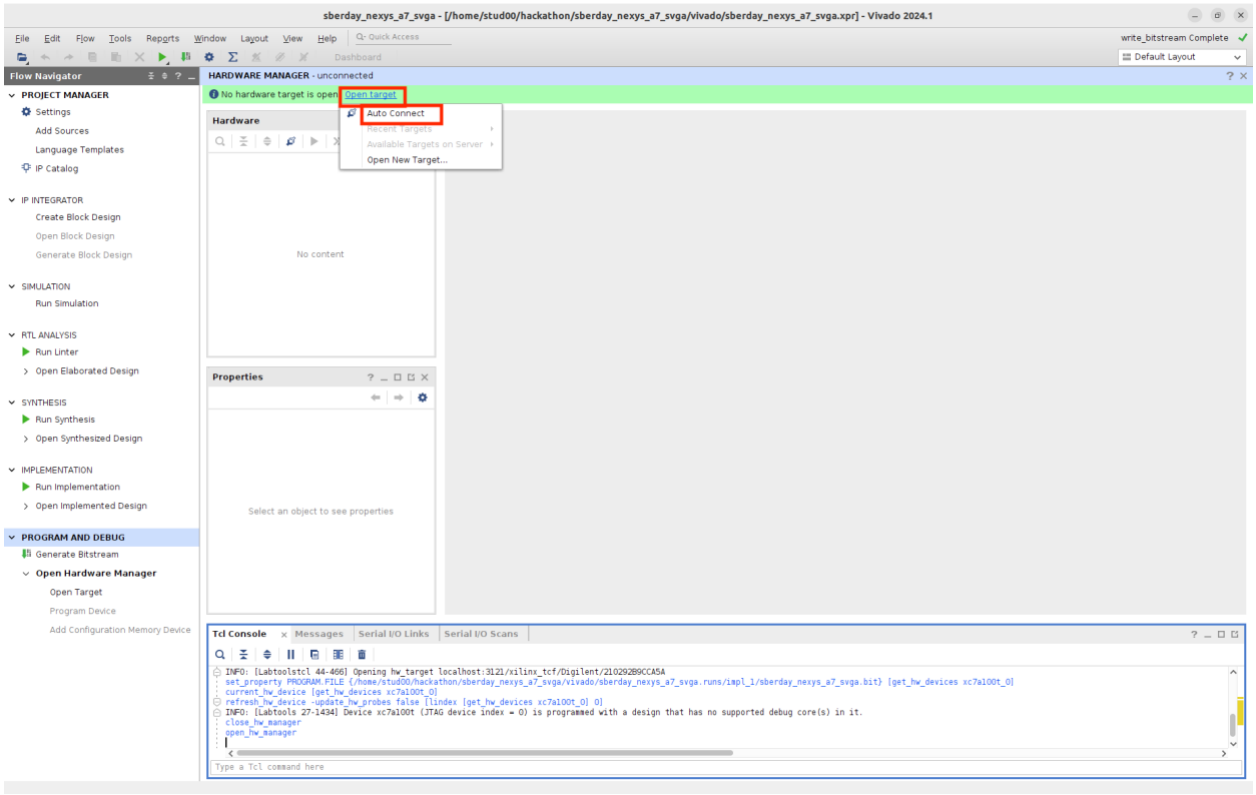
Когда все будет выполнено, Vivado сообщит об этом как показано на рисунке ниже. Выберите пункт *«Open Hardware Manager»*. Если вы по какой-либопричине не выбрали этот пункт, то можно открыть Hardware Manager кнопкой под *«Generate bitstream»*, в меню Project Navigator слева, подробнее это описано в инструкции по использованию Vivado в репозитории к проекту.



Уведомление о том, что битстрим сгенерирован.

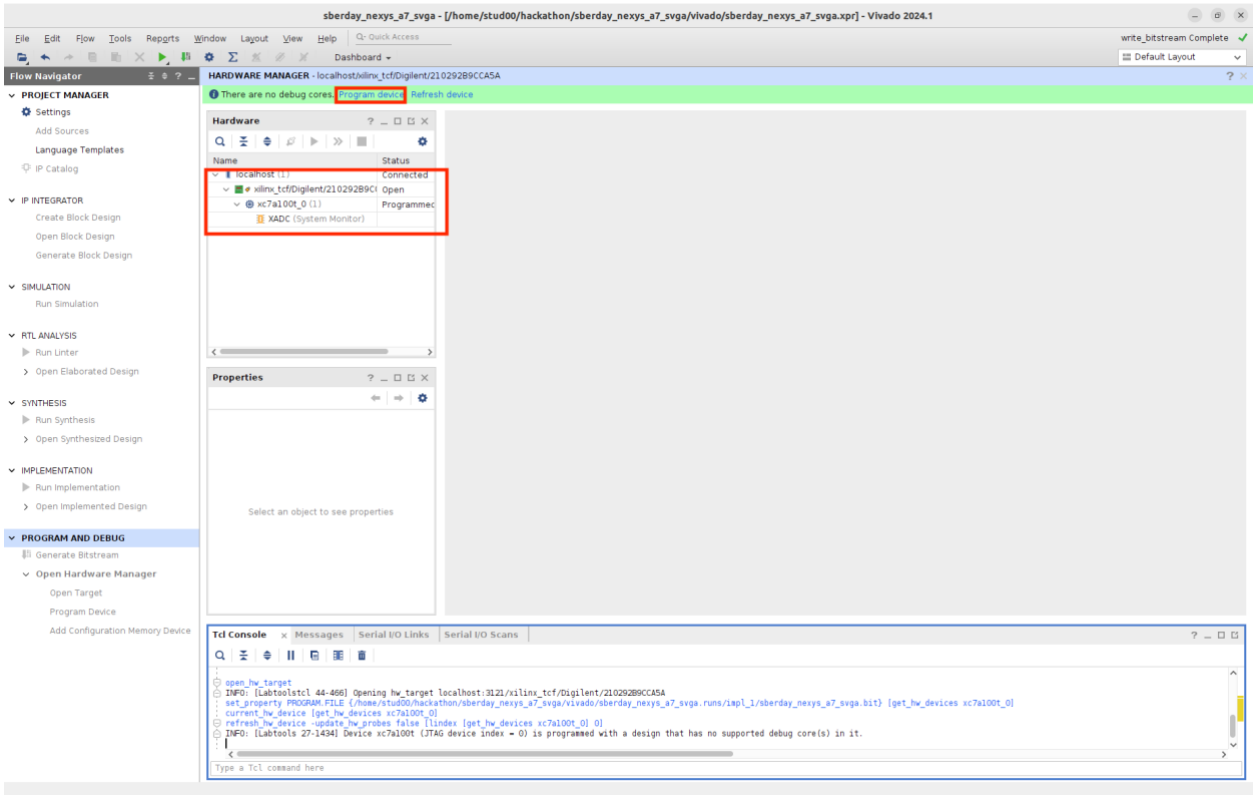
3. Прошивка платы и настройка дисплея

После открытия *«Hardware manager»*, нужно подключиться к отладочной плате, для этого нажмите *«Open Target»*, затем *«Auto Connect»* как показано на рисунке ниже.



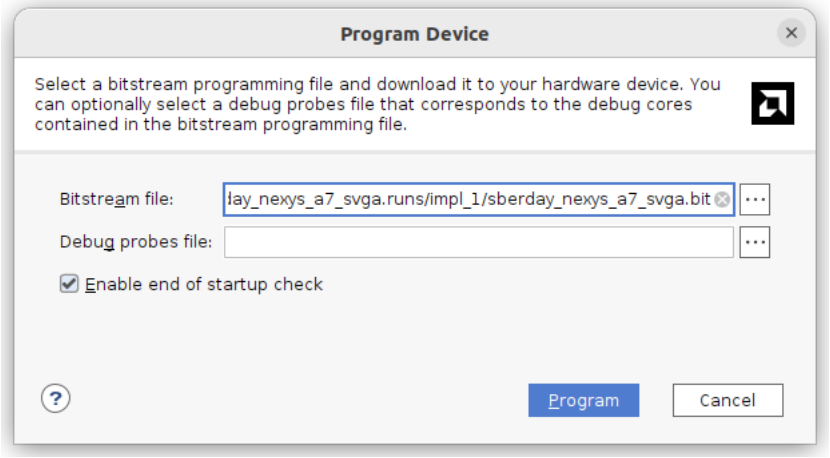
Подключение Vivado к отладочной плате.

Когда плата определится средой, в окне *«Hardware»* появится информация об используемой ПЛИС как показано на рисунке ниже. Далее нажмите *«Program Device»*.



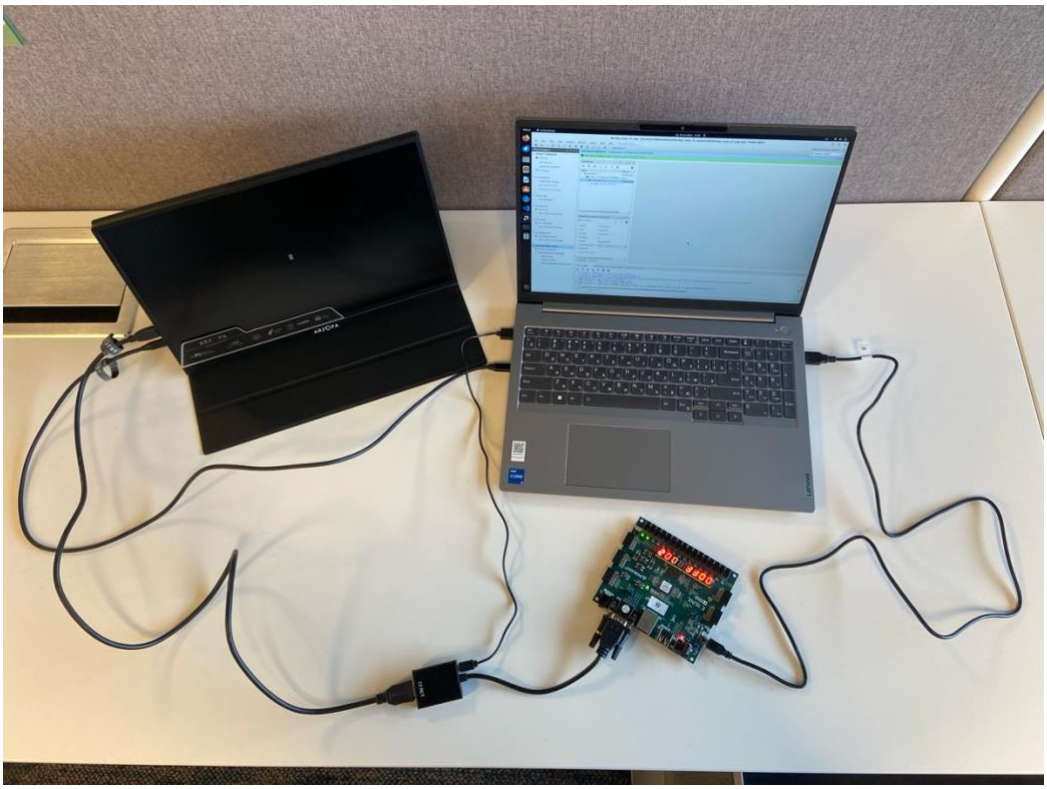
Загрузка битстрима в ПЛИС.

Появится окно для выбора *.bit файла, путь до сгенерированного ранее в рамках проекта файла будет выбран автоматически. Нажмите на кнопку *«Program»*.



Выбор битстрима для загрузки.

После появится индикатор загрузки. На этом прошивка платы завершена. При перепрошивке платы пропадает VGA сигнал от платы и дисплей может снова перейти в режим работы как дополнительный дисплей ноутбука, в таком случае вам потребуется снова настроить его на источник изображения HDMI как описывалось выше.
Также учитывайте, что при выключении питания платы, загруженная прошивка будет сброшена и при включении питания произойдет загрузка заводской прошивки из памяти платы.
Если все прошло успешно, вы увидите сначала логотип Сбера, а после белый квадрат на дисплее как показано на рисунке ниже.



Результат запуска демо.